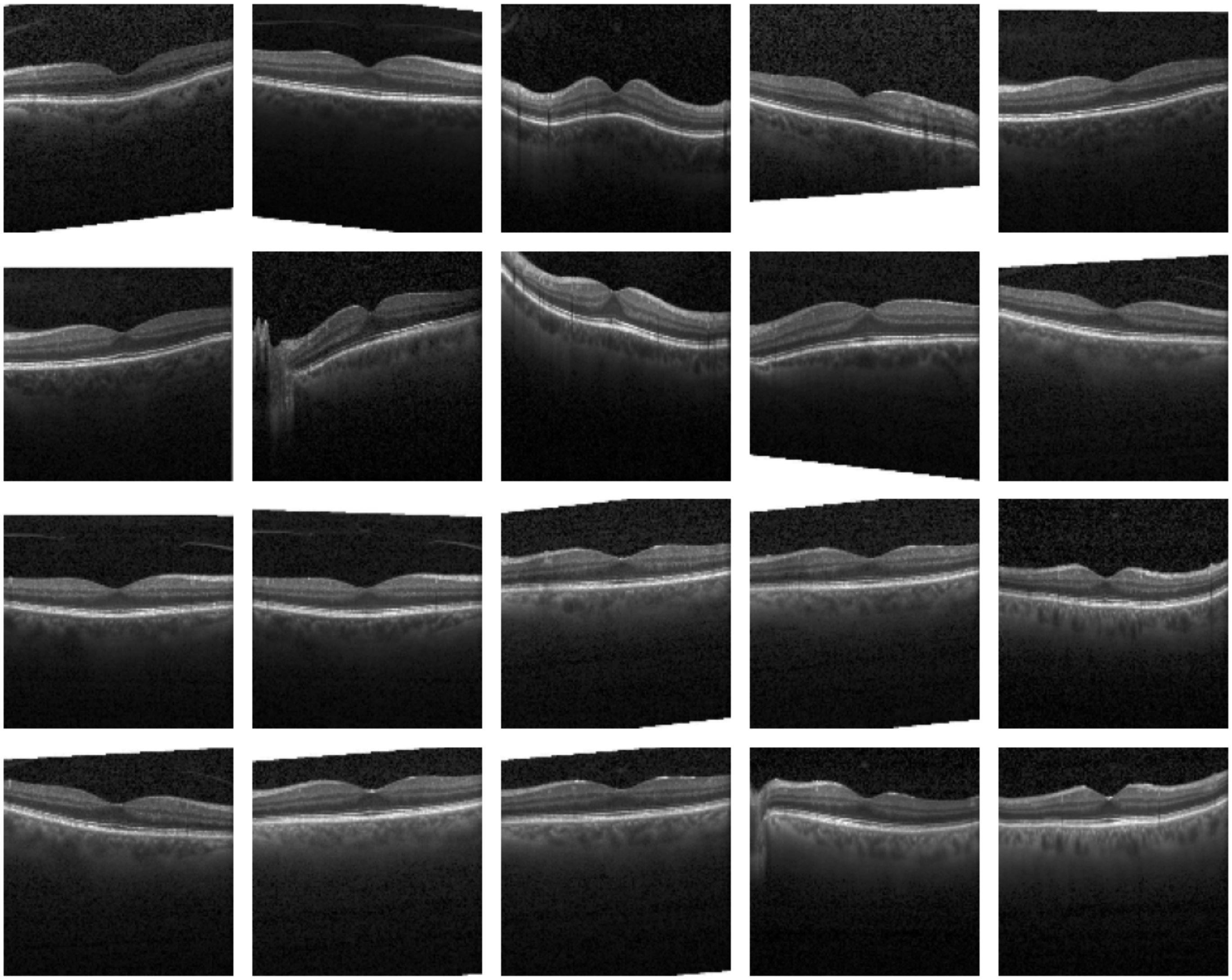
In [1]:
```
!python --version
```

```
Python 3.11.5
```

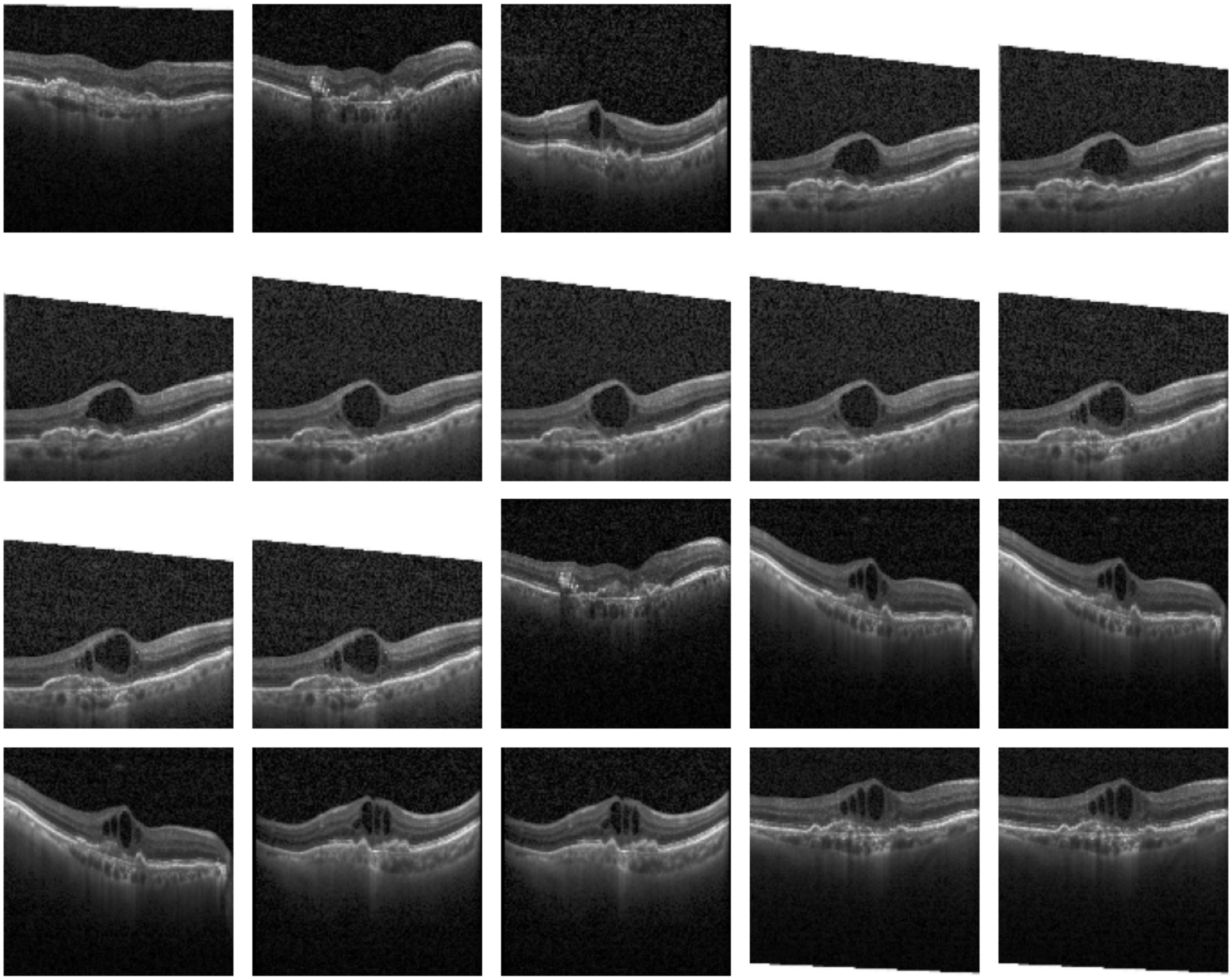
In [2]:
```
from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.autograd import Variable
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import cv2
import time
import os
import copy
import skimage
from tensorflow.keras.utils import to_categorical

imageSize=200
train_dir = "C:/Users/sumed/Downloads/archive (12)/OCT2017/train/"
test_dir =  "C:/Users/sumed/Downloads/archive (12)/OCT2017/test/"
val_dir =  "C:/Users/sumed/Downloads/archive (12)/OCT2017/val/"
# ['DME', 'CNV', 'NORMAL', '.DS_Store', 'DRUSEN']
from tqdm import tqdm
```

In [3]:
```
from pathlib import Path
normal_dir = Path("C:/Users/sumed/Downloads/archive (12)/OCT2017/train/NORMAL")

# Gather all image file paths and select the first 20
image_paths = list(normal_dir.glob('*'))[:20]

# Set up the plotting parameters
plt.figure(figsize=(10, 8))  # Adjust size as needed
for i, image_path in enumerate(image_paths):
    # Read and resize each image
    image = cv2.imread(str(image_path))
    image = cv2.resize(image, (128, 128))
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert color format

    # Plot each image in a 4x5 grid
    plt.subplot(4, 5, i + 1)
    plt.imshow(image)
    plt.axis('off')  # Hide axes for a cleaner display

plt.tight_layout()  # Adjust layout to fit images well
plt.show()
```



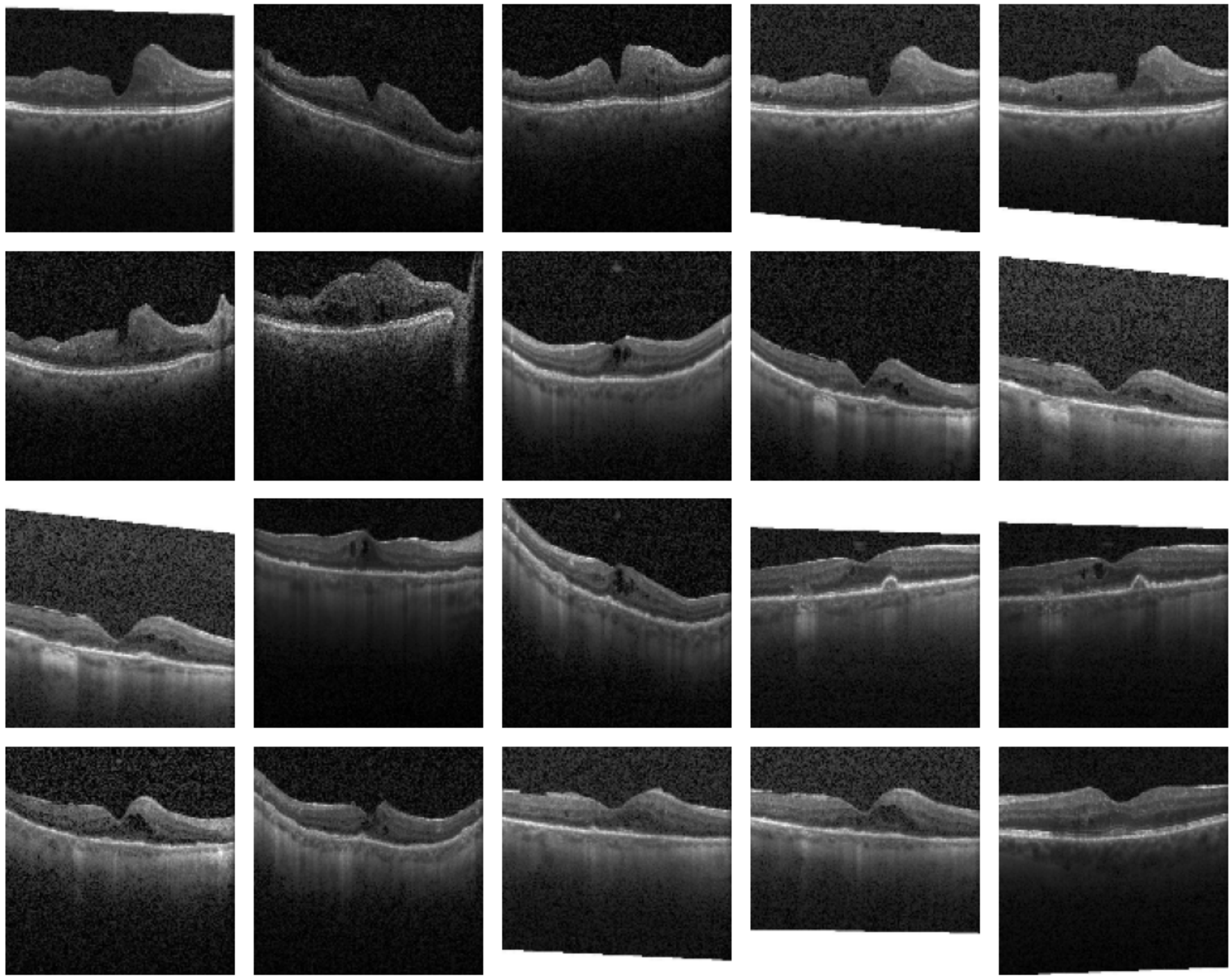
In [4]:

```
cnv_dir = Path("C:/Users/sumed/Downloads/archive (12)/OCT2017/train/CNV")

# Gather all image file paths and select the first 20
image_paths_cnv = list(cnv_dir.glob('*'))[:20]

# Set up the plotting parameters
plt.figure(figsize=(10, 8))  # Adjust size as needed
for i, image_path in enumerate(image_paths_cnv):
    # Read and resize each image
    image_cnv = cv2.imread(str(image_path))  # Use the correct image path here
    if image_cnv is None:
        print(f"Failed to load image: {image_path}")
        continue  # Skip this image if it can't be loaded
    image_cnv = cv2.resize(image_cnv, (128, 128))  # Resize the image
    image_cnv = cv2.cvtColor(image_cnv, cv2.COLOR_BGR2RGB)  # Convert color format

    # Plot each image in a 4x5 grid
    plt.subplot(4, 5, i + 1)
    plt.imshow(image_cnv)
    plt.axis('off')  # Hide axes for a cleaner display

plt.tight_layout()  # Adjust layout to fit images well
plt.show()
```



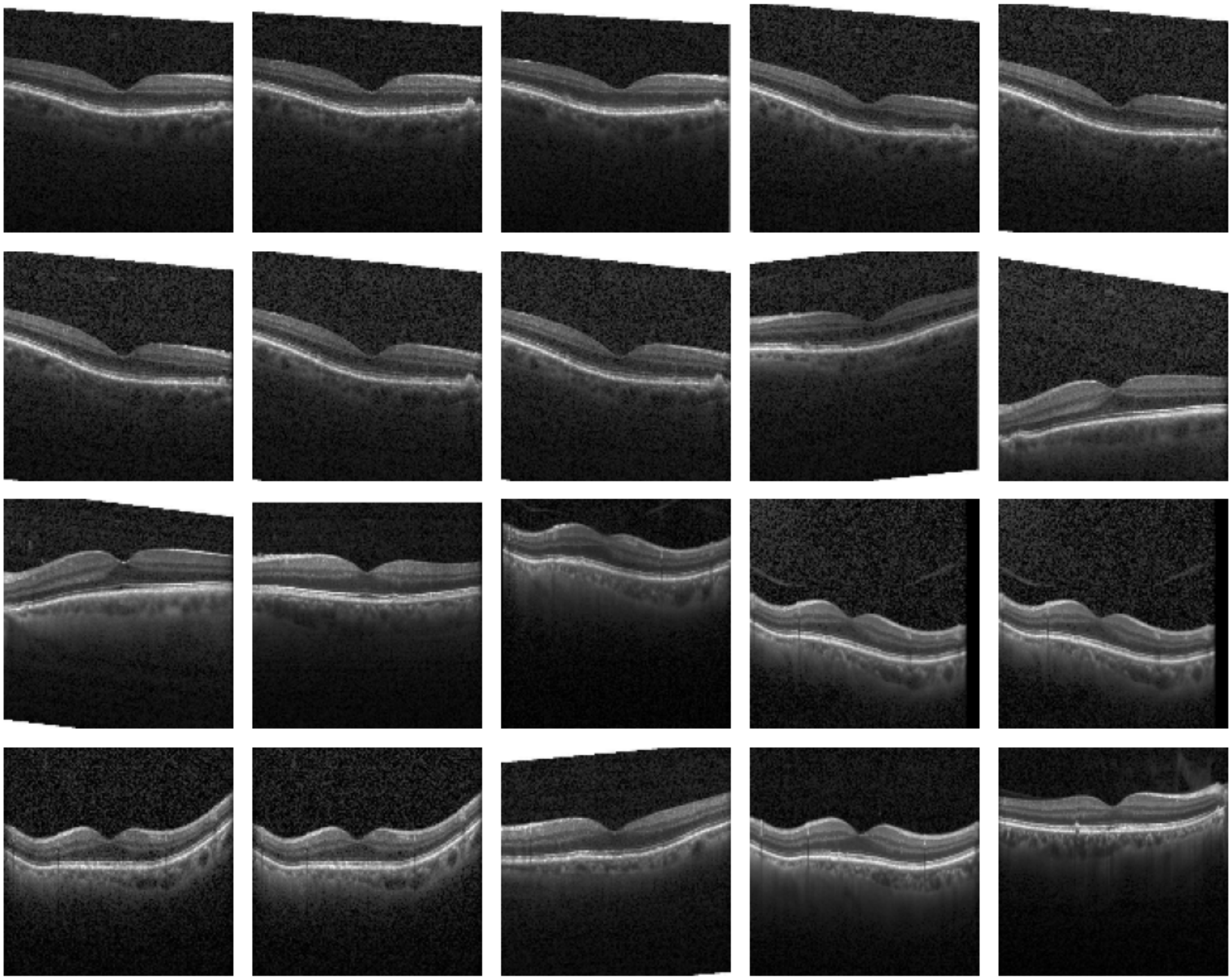
In [5]:

```python
dme_dir = Path("C:/Users/sumed/Downloads/archive (12)/OCT2017/train/DME")

# Gather all image file paths and select the first 20
image_paths_dme = list(dme_dir.glob('*'))[:20]

# Set up the plotting parameters
plt.figure(figsize=(10, 8))  # Adjust size as needed
for i, image_path in enumerate(image_paths_dme):
    # Read and resize each image
    image_dme = cv2.imread(str(image_path))

    if image_dme is None:
        print(f"Failed to load image: {image_path}")
        continue  # Skip this image if it can't be loaded

    image_dme = cv2.resize(image_dme, (128, 128))  # Resize the image
    image_dme = cv2.cvtColor(image_dme, cv2.COLOR_BGR2RGB)  # Convert color format

    # Plot each image in a 4x5 grid
    plt.subplot(4, 5, i + 1)
    plt.imshow(image_dme)
    plt.axis('off')  # Hide axes for a cleaner display

plt.tight_layout()  # Adjust layout to fit images well
plt.show()
```



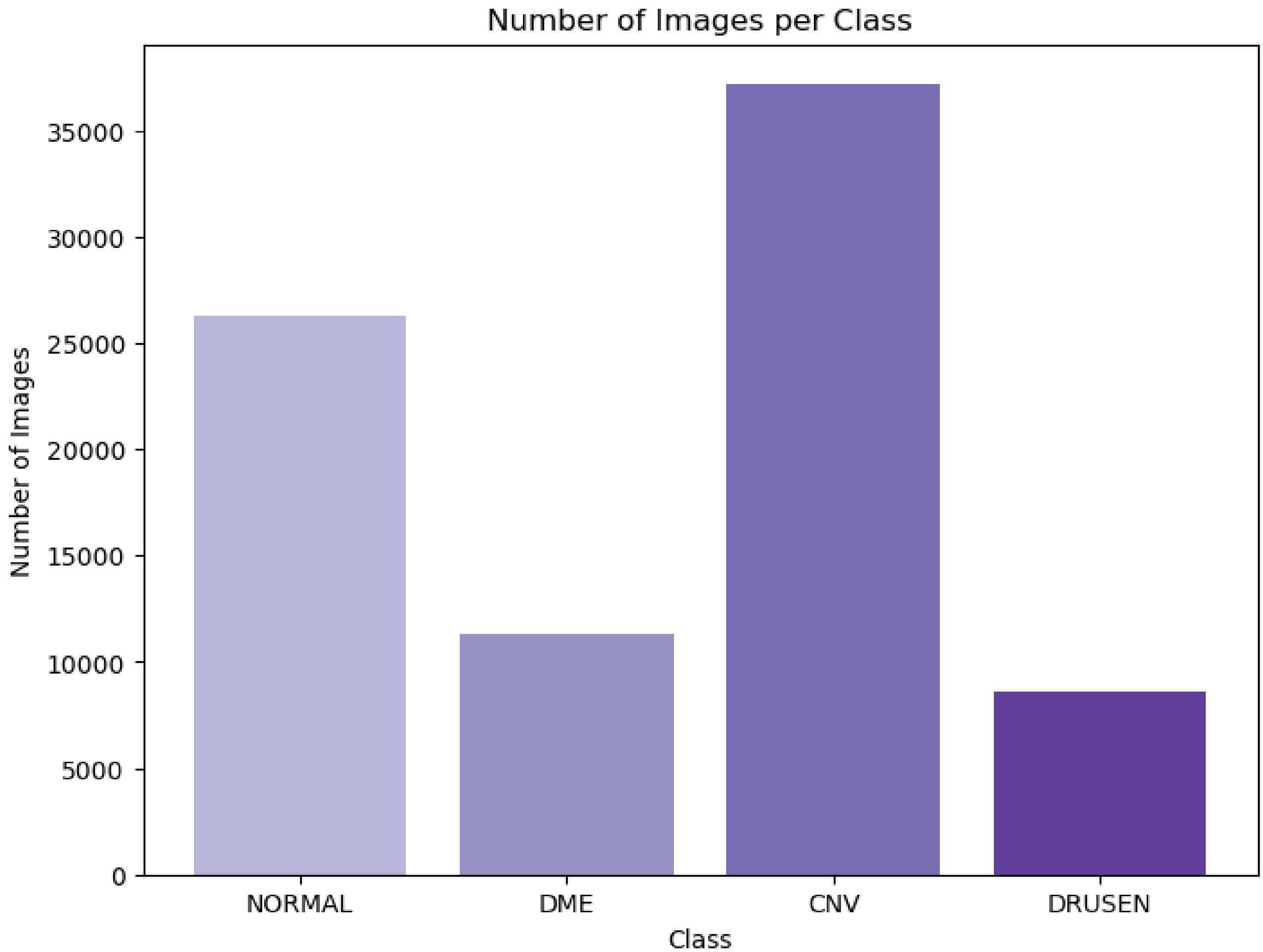
In [6]:

```
drusen_dir = Path("C:/Users/sumed/Downloads/archive (12)/OCT2017/train/DRUSEN")

# Gather all image file paths and select the first 20
image_paths_drusen = list(drusen_dir.glob('*'))[:20]

# Set up the plotting parameters
plt.figure(figsize=(10, 8))  # Adjust size as needed
for i, image_path in enumerate(image_paths_drusen):
    # Read and resize each image
    image_drusen = cv2.imread(str(image_path))

    if image_drusen is None:
        print(f"Failed to load image: {image_path}")
        continue  # Skip this image if it can't be loaded

    image_drusen = cv2.resize(image_drusen, (128, 128))  # Resize the image
    image_drusen = cv2.cvtColor(image_drusen, cv2.COLOR_BGR2RGB)  # Convert color format

    # Plot each image in a 4x5 grid
    plt.subplot(4, 5, i + 1)
    plt.imshow(image_drusen)
    plt.axis('off')  # Hide axes for a cleaner display

plt.tight_layout()  # Adjust layout to fit images well
plt.show()
```



In [7]:

```
# Define the directories for each class
class_dirs = {
    'NORMAL': Path("C:/Users/sumed/Downloads/archive (12)/OCT2017/train/NORMAL"),
    'DME': Path("C:/Users/sumed/Downloads/archive (12)/OCT2017/train/DME"),
    'CNV': Path("C:/Users/sumed/Downloads/archive (12)/OCT2017/train/CNV"),
    'DRUSEN': Path("C:/Users/sumed/Downloads/archive (12)/OCT2017/train/DRUSEN")
}

# Initialize a dictionary to store the counts
class_counts = {}

# Loop through each class directory and count the images
for class_name, class_dir in class_dirs.items():
    class_counts[class_name] = len(list(class_dir.glob('*')))  # Count files in each cla

# Plot the results in a bar graph
plt.figure(figsize=(8, 6))
colors = plt.cm.Purples(np.linspace(0.4, 0.8, len(class_counts)))

plt.bar(class_counts.keys(), class_counts.values(), color=colors)

# Add labels and title
plt.xlabel('Class')
plt.ylabel('Number of Images')
plt.title('Number of Images per Class')
plt.show()
```



The weights are computed to make the model focus more on the classes that are less frequent. Since the weights are close to 1, it suggests that your dataset does not have a severe class imbalance. The model will treat DME (which appears to have slightly more samples) with a slightly lower weight, and NORMAL, CNV, and DRUSEN will get roughly equal treatment with weights near 1.

In [8]:
```
img_size = (128, 128)
batch_size = 32
```

In [9]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=10, width_shift_range=
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

In [10]:
```
train_generator = train_datagen.flow_from_directory(train_dir, target_size=img_size, bat
val_generator = val_datagen.flow_from_directory(val_dir, target_size=img_size, batch_siz
test_generator = test_datagen.flow_from_directory(test_dir, target_size=img_size, batch_
```

```
Found 83484 images belonging to 4 classes.
Found 32 images belonging to 4 classes.
Found 968 images belonging to 4 classes.
```

In [11]:
```
from sklearn.utils.class_weight import compute_class_weight

# Extract class indices and their corresponding counts from the training generator
class_indices = train_generator.class_indices
classes = list(class_indices.keys())  # Class names
y_train = train_generator.classes      # True class labels from the generator

# Compute class weights
```

```python
class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)

# Convert to dictionary format for easy mapping to class names
class_weights_dict = dict(zip(class_indices.values(), class_weights))

print(f"Class Weights: {class_weights_dict}")
```

```
Class Weights: {0: 0.5609729875016799, 1: 1.8391787099048291, 2: 2.4223537604456826, 3: 0.7931217936538096}
```

In [11]:

```python
import tensorflow as tf

# Build CNN model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4, activation='softmax')
])
```

```
C:\Users\sumed\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [12]:

```python
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [13]:

```python
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516

Total params: 3,305,156 (12.61 MB)

Trainable params: 3,305,156 (12.61 MB)

Non-trainable params: 0 (0.00 B)

In [16]:
```
model_training = model.fit(train_generator, validation_data=val_generator, epochs=10)
```

```
Epoch 1/10
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_adap
ter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)`
in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_
size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 1929s 737ms/step - accuracy: 0.6299 - loss:
 0.9498 - val_accuracy: 0.5312 - val_loss: 1.0733
Epoch 2/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 1819s 696ms/step - accuracy: 0.7601 - loss:
 0.6444 - val_accuracy: 0.8438 - val_loss: 0.5039
Epoch 3/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 1776s 680ms/step - accuracy: 0.8437 - loss:
 0.4470 - val_accuracy: 0.8125 - val_loss: 0.5071
Epoch 4/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 1716s 657ms/step - accuracy: 0.8739 - loss:
 0.3688 - val_accuracy: 0.9688 - val_loss: 0.0807
Epoch 5/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 1809s 675ms/step - accuracy: 0.8839 - loss:
 0.3461 - val_accuracy: 0.9375 - val_loss: 0.1014
Epoch 6/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 1680s 643ms/step - accuracy: 0.8956 - loss:
 0.3128 - val_accuracy: 1.0000 - val_loss: 0.0518
Epoch 7/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 1708s 645ms/step - accuracy: 0.9015 - loss:
 0.2974 - val_accuracy: 1.0000 - val_loss: 0.0556
Epoch 8/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 1674s 641ms/step - accuracy: 0.9064 - loss:
 0.2811 - val_accuracy: 0.9375 - val_loss: 0.0938
Epoch 9/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 1676s 642ms/step - accuracy: 0.9084 - loss:
 0.2800 - val_accuracy: 0.9688 - val_loss: 0.0318
Epoch 10/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 1705s 643ms/step - accuracy: 0.9104 - loss:
 0.2703 - val_accuracy: 0.9688 - val_loss: 0.0477
```

In [18]:
```
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

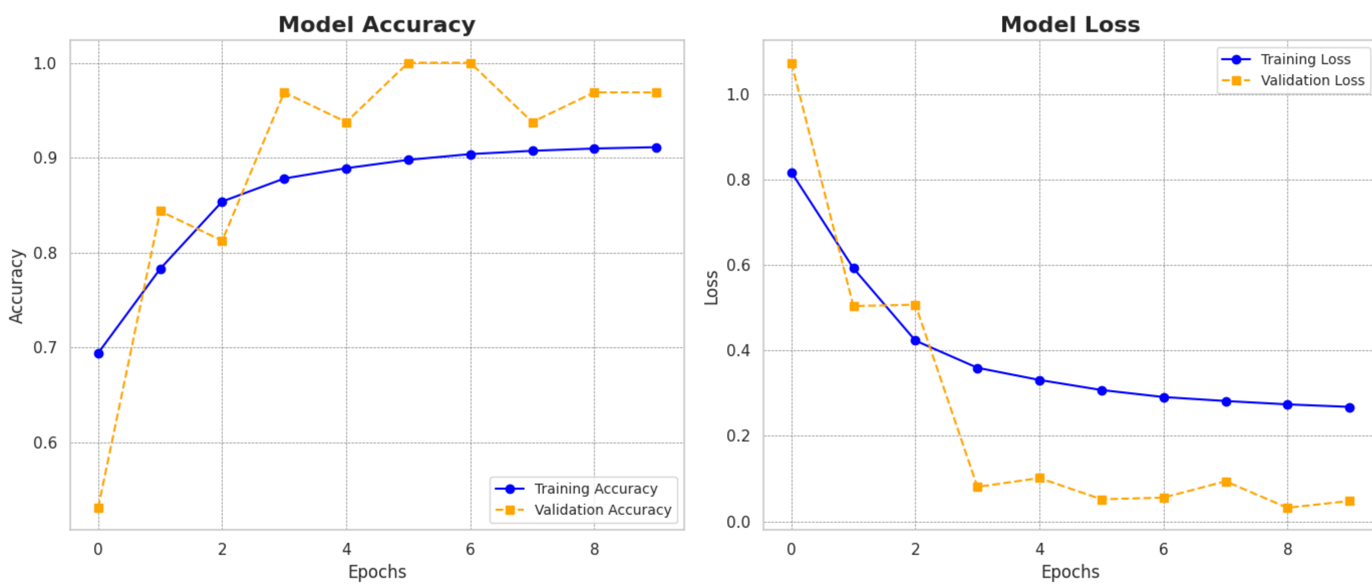
```
31/31 ━━━━━━━━━━━━━━━━━━━━ 10s 316ms/step - accuracy: 0.9712 - loss: 0.116
4
Test Loss: 0.1068936362862587
Test Accuracy: 97.21%
```

In [19]:
```
# Predicting on the test set
test_predictions = model.predict(test_generator)
test_predictions_labels = np.argmax(test_predictions, axis=1)
true_labels = test_generator.classes
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 7s 205ms/step
```

In [22]:
```
import seaborn as sns

# Plot training and validation accuracy
sns.set_theme(style="whitegrid")

# Extract training history
history = model_training.history
```

```python
# Create subplots for accuracy and loss
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Plot training and validation accuracy
axes[0].plot(history['accuracy'], label='Training Accuracy', marker='o', linestyle='-',
axes[0].plot(history['val_accuracy'], label='Validation Accuracy', marker='s', linestyle
axes[0].set_title('Model Accuracy', fontsize=16, weight='bold')
axes[0].set_xlabel('Epochs', fontsize=12)
axes[0].set_ylabel('Accuracy', fontsize=12)
axes[0].legend(loc='lower right', fontsize=10)
axes[0].grid(color='gray', linestyle='--', linewidth=0.5)

# Plot training and validation loss
axes[1].plot(history['loss'], label='Training Loss', marker='o', linestyle='-', color='b
axes[1].plot(history['val_loss'], label='Validation Loss', marker='s', linestyle='--', c
axes[1].set_title('Model Loss', fontsize=16, weight='bold')
axes[1].set_xlabel('Epochs', fontsize=12)
axes[1].set_ylabel('Loss', fontsize=12)
axes[1].legend(loc='upper right', fontsize=10)
axes[1].grid(color='gray', linestyle='--', linewidth=0.5)

# Tight layout for better spacing
plt.tight_layout()

# Show the plots
plt.show()
```



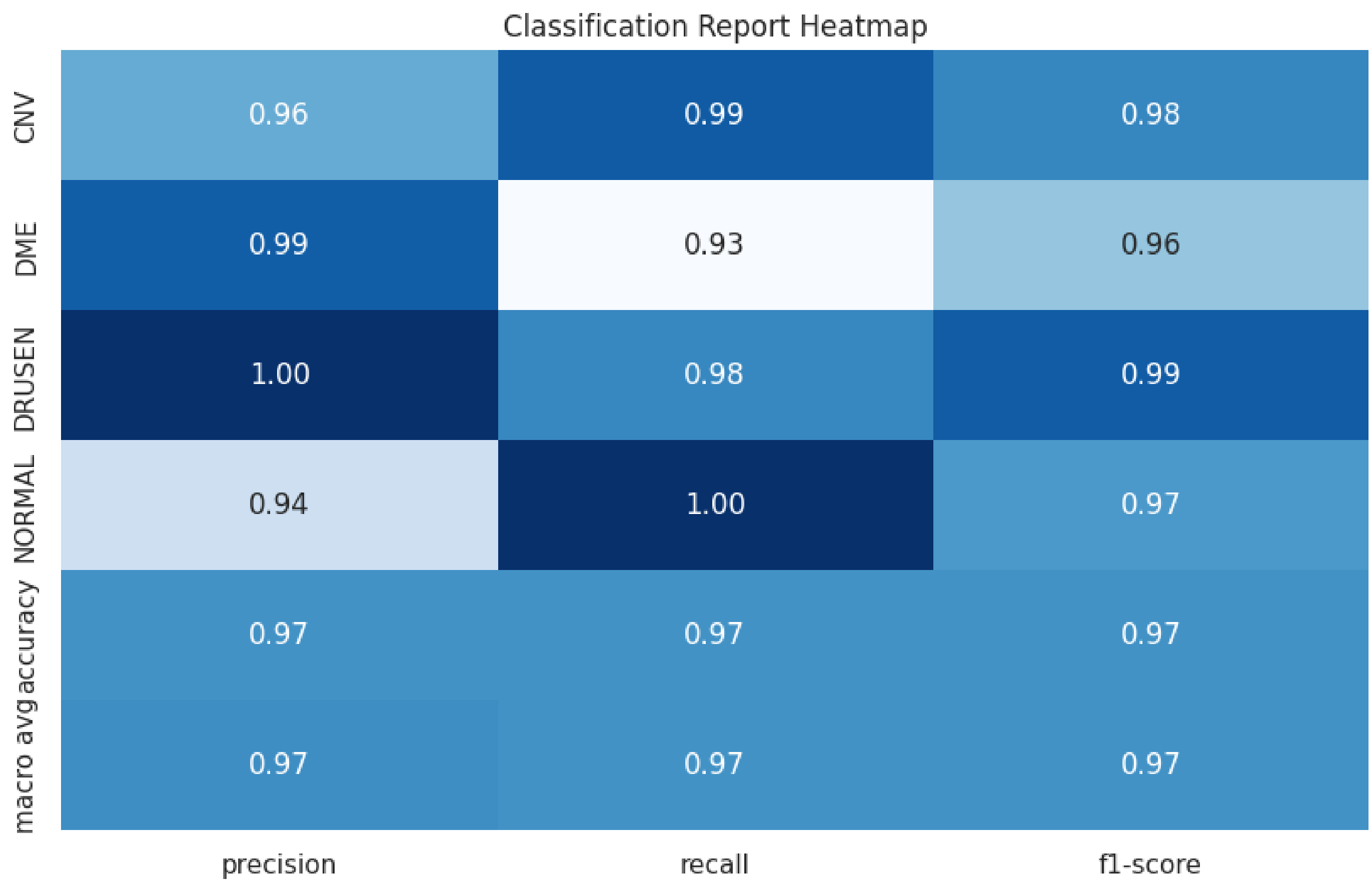
In [23]:
```python
from sklearn.metrics import classification_report

# Generate predictions on the test set
y_pred = model.predict(test_generator)
y_pred_classes = np.argmax(y_pred, axis=1)  # Convert probabilities to class indices
y_true = test_generator.classes  # True class labels

# Class labels (assuming the generator's classes are in order of the directories)
class_labels = list(test_generator.class_indices.keys())

# Create the classification report
report = classification_report(y_true, y_pred_classes, target_names=class_labels)

# Print the report
print("Classification Report:")
print(report)
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 7s 209ms/step
Classification Report:
```

```
              precision    recall  f1-score   support

         CNV       0.96      0.99      0.98       242
         DME       0.99      0.93      0.96       242
      DRUSEN       1.00      0.98      0.99       242
      NORMAL       0.94      1.00      0.97       242

    accuracy                           0.97       968
   macro avg       0.97      0.97      0.97       968
weighted avg       0.97      0.97      0.97       968
```

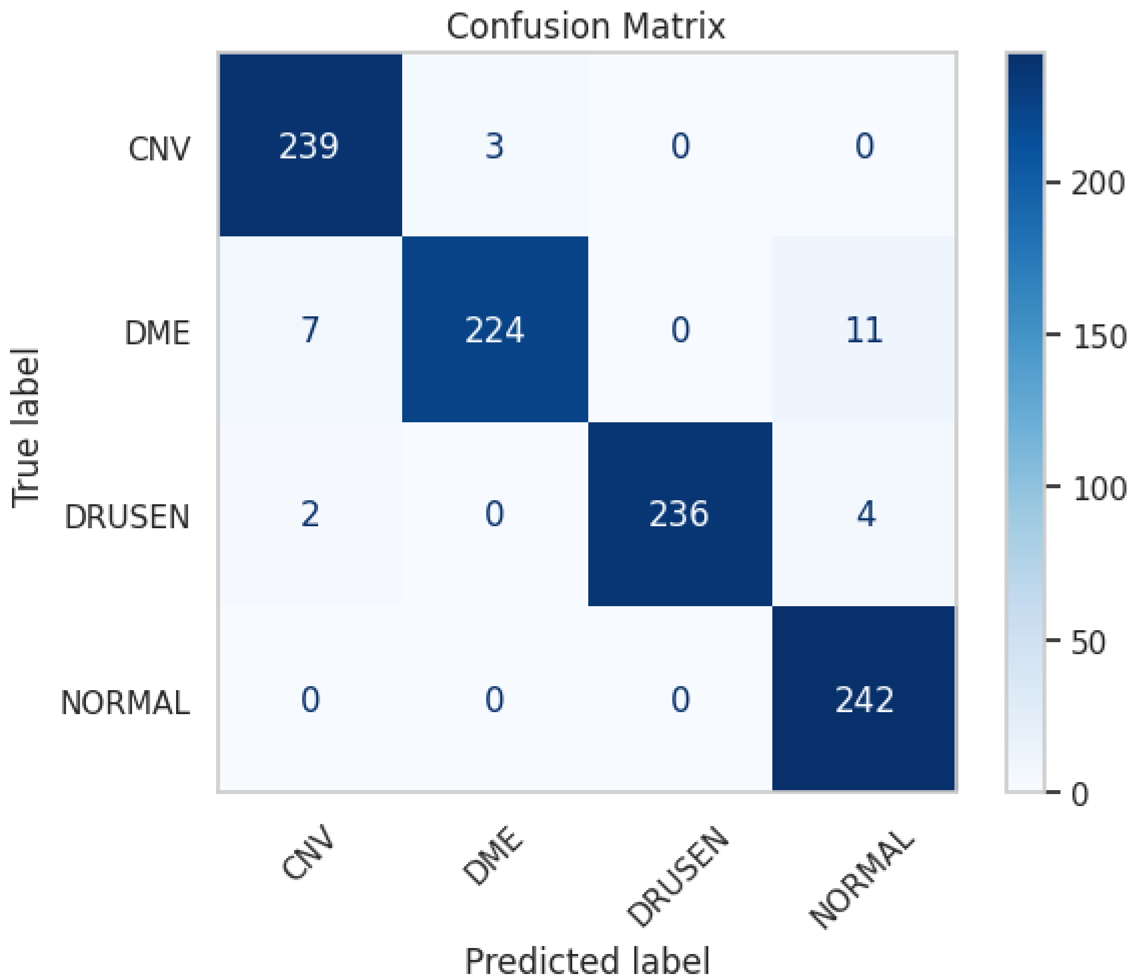
In [24]:

```python
import pandas as pd

# Convert the classification report to a dataframe
report_dict = classification_report(y_true, y_pred_classes, target_names=class_labels, o
report_df = pd.DataFrame(report_dict).transpose()

# Plot the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(report_df.iloc[:-1, :-1], annot=True, fmt=".2f", cmap="Blues", cbar=False)
plt.title("Classification Report Heatmap")
plt.show()
```



In [27]:

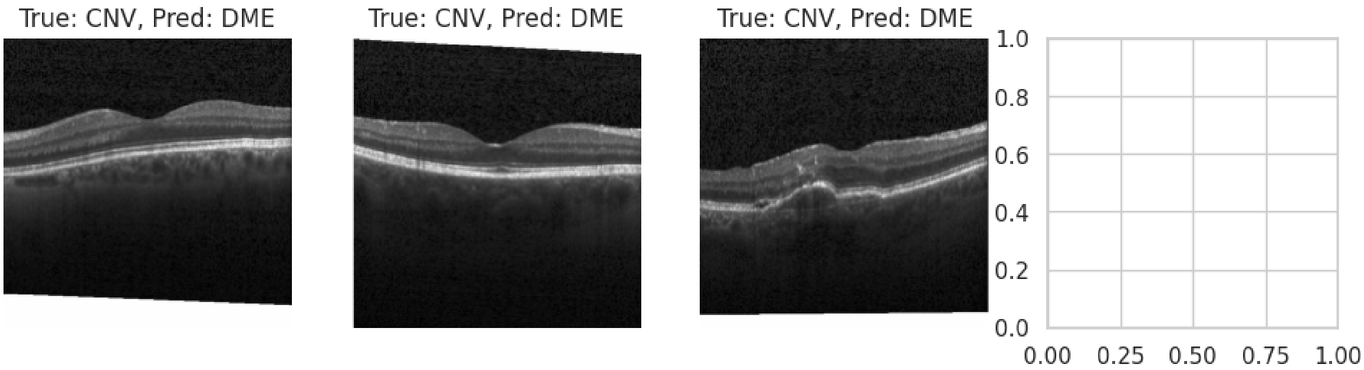
```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Generate confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
disp.plot(cmap='Blues', xticks_rotation=45)
plt.title("Confusion Matrix")
plt.grid(False)
plt.show()
```



In [37]:

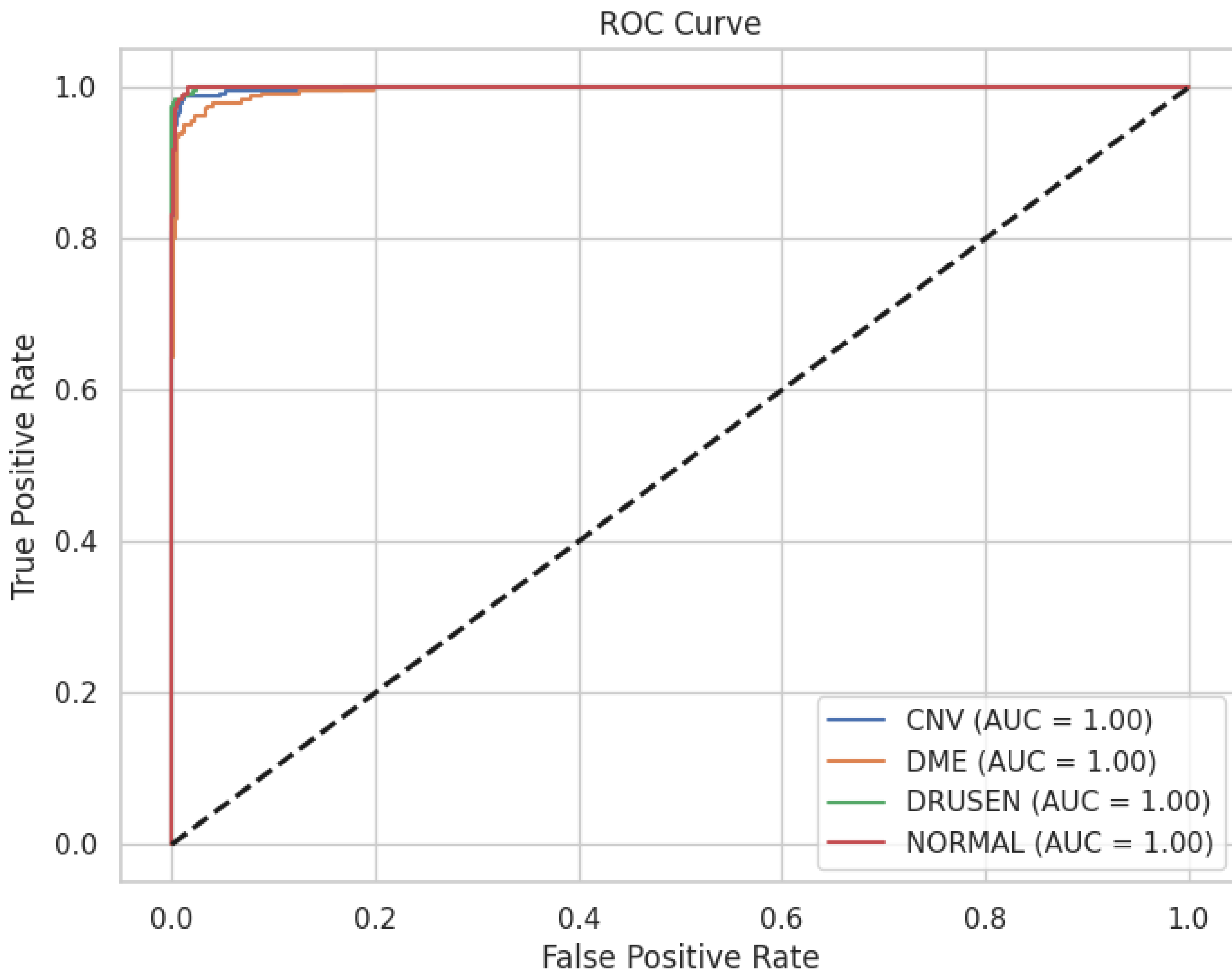
```python
# Identify misclassified indices
misclassified_indices = np.where(y_true != y_pred_classes)[0]

# Plot misclassified images
plt.figure(figsize=(12, 12))
for i, idx in enumerate(misclassified_indices[:15]):  # First 16 misclassified images
    plt.subplot(4, 4, i+1)
    plt.imshow(X_test[idx])
    plt.title(f"True: {class_labels[y_true[idx]]}, Pred: {class_labels[y_pred_classes[id
    plt.axis('off')
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[37], line 8
      6 for i, idx in enumerate(misclassified_indices[:15]):  # First 16 misclassified i
mages
      7     plt.subplot(4, 4, i+1)
----> 8     plt.imshow(X_test[idx])
      9     plt.title(f"True: {class_labels[y_true[idx]]}, Pred: {class_labels[y_pred_cl
asses[idx]]}")
     10     plt.axis('off')

IndexError: index 287 is out of bounds for axis 0 with size 194
```



In [34]:

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

# Binarize the labels for multi-class ROC
y_true_bin = label_binarize(y_true, classes=np.arange(len(class_labels)))

# Compute ROC curve and AUC for each class
fpr = {}
tpr = {}
roc_auc = {}
for i in range(len(class_labels)):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure(figsize=(8, 6))
for i, label in enumerate(class_labels):
    plt.plot(fpr[i], tpr[i], label=f'{label} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.show()
```



In [14]:
```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [15]:
```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

# Freeze the base model layers to retain pre-trained weights
base_model.trainable = True
```

In [16]:
```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
output = Dense(4, activation='softmax')(x)

# Final model
model_resnet = Model(inputs=base_model.input, outputs=output)

# Compile the model
model_resnet.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [16]:
```
history = model_resnet.fit(
    train_generator,
    validation_data=val_generator,
    epochs=10,
```

```
    verbose=1
)
```

```
Epoch 1/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 9054s 3s/step - accuracy: 0.8364 - loss: 0.
5201 - val_accuracy: 0.9062 - val_loss: 0.3096
Epoch 2/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 8016s 3s/step - accuracy: 0.9078 - loss: 0.
2855 - val_accuracy: 1.0000 - val_loss: 0.0427
Epoch 3/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 8003s 3s/step - accuracy: 0.9083 - loss: 0.
2776 - val_accuracy: 0.9688 - val_loss: 0.0509
Epoch 4/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 8006s 3s/step - accuracy: 0.9263 - loss: 0.
2262 - val_accuracy: 0.9375 - val_loss: 0.1253
Epoch 5/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 7849s 3s/step - accuracy: 0.9214 - loss: 0.
2437 - val_accuracy: 1.0000 - val_loss: 0.0874
Epoch 6/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 7977s 3s/step - accuracy: 0.9277 - loss: 0.
2170 - val_accuracy: 1.0000 - val_loss: 0.0111
Epoch 7/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 7809s 3s/step - accuracy: 0.9289 - loss: 0.
2215 - val_accuracy: 1.0000 - val_loss: 0.0159
Epoch 8/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 8031s 3s/step - accuracy: 0.9351 - loss: 0.
1952 - val_accuracy: 1.0000 - val_loss: 0.0150
Epoch 9/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 7957s 3s/step - accuracy: 0.9286 - loss: 0.
2204 - val_accuracy: 1.0000 - val_loss: 0.0303
Epoch 10/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 7757s 3s/step - accuracy: 0.9376 - loss: 0.
1862 - val_accuracy: 1.0000 - val_loss: 0.0115
```

In [17]:

```
test_loss, test_accuracy = model_resnet.evaluate(test_generator)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_accuracy}")
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 30s 978ms/step - accuracy: 0.9934 - loss: 0.033
0
Test Loss: 0.031029945239424706, Test Accuracy: 0.9927685856819153
```